University of L'Aquila, Degree in Computer Science
Course on Formal Methods 2009-2010

Lecture Notes on

First-Order Term Rewriting Systems

Preliminary version

Monica Nesi

June 2010

These lecture notes contain an English version of *some* sections taken from the technical report "Sistemi di Riscrittura per Termini del Prim'Ordine" by Paola Inverardi, Monica Nesi and Marisa Venturini Zilli, T.R. No. 35, 1999, Dip. Matematica Pura e Applicata, Università degli Studi di L'Aquila.
The translated sections include most of the contents of the course on rewriting given in the module on *Formal Methods* in the academic year 2009-10.

Index

1 Introduction **1**

2 First-order terms **1**
2.1 Term representation as labelled ordered trees 2
2.2 Subterms, occurrences, positions, contexts 3
2.3 Substitutions 5
2.4 Matching substitutions and unifiers 6
2.5 Algorithms of symbolic manipulation on terms 7
2.5.1 Instantiation 7
2.5.2 Unification 8
2.5.3 Anti-unification 10

3 Equational calculus **10**
3.1 Equational deduction 11
3.2 Models 13

4 Term rewriting systems **13**
4.1 Redex, derivation, normal form 14

5 Terminating rewrite systems **15**
5.1 Partial orderings useful for termination 16
5.1.1 Reduction orderings 16
5.1.2 Simplification orderings 18
5.1.3 Recursive path orderings 19

6 Confluent rewrite systems **21**
6.1 Spectrum of the rules of a rewrite system 21
6.2 Critical pairs 22
6.3 The Huet Lemma 25

7 Canonical rewrite systems **27**
7.1 The word problem 27
7.2 The Knuth-Bendix Theorem 28
7.3 Completion procedures 28
7.4 Properties of the completion rules 31

8 Rewriting modulo equations **32**
8.1 E-unification 32
8.2 Narrowing 34
8.3 The E-unification procedure 35
8.4 Equational theories and E-unification: a summary 38
8.5 Comparison with SLD-resolution 40

1 Introduction

A *term rewriting system* (or simply *rewrite system*) is a particular abstract reduction system $\langle A, \longrightarrow \rangle$ [12], where A is the set $\mathcal{T}(\Sigma, V)$ of first-order terms with function symbols in Σ and variables in V.

In the following some basic notions on term rewriting will be introduced. In particular, we will consider the completion of an equational theory E and the E-unification of terms. For more details the reader can refer to [1, 3].

2 First-order terms

Let us define the set of terms in a first-order language without *sorts* (the so-called *unsorted* terms), as this restriction simplifies the formalization while preserving all interesting notions.

Given a set of function symbols Σ, called *signature*, the terms on such a signature are defined as all well-formed expressions that can be built starting from the function symbols in Σ. If we also consider a countable set of variables V, we obtain the set of terms over Σ and with variables in V.
The number $n \in \mathbf{N}$ of the arguments of a function symbol $f \in \Sigma$ is called the *arity* of f. If $n = 0$, f is a *constant* symbol. Σ^n denotes the set of all function symbols in Σ with arity n. Thus, we have $\Sigma = \bigcup_n \Sigma^n$.

Definition 1 *Let Σ be a finite signature and V be a countable set of variables. The set $Ter = \mathcal{T}(\Sigma, V)$ of (finite) terms with function symbols in Σ and variables in V is defined as follows:*

i) $f \in Ter$ *for all* $f \in \Sigma^0$
ii) $V \subseteq Ter$
iii) $f(t_1, \ldots, t_n) \in Ter$ *for all* $f \in \Sigma^n$ *and* $t_i \in Ter$, $i = 1, \ldots, n$.

Equality in Ter, written $=$, is the syntactic identity.
In the following we will indifferently use Ter, $\mathcal{T}(\Sigma, V)$ or $Ter(\Sigma, V)$ for denoting the set of terms over a signature Σ and with variables in V. $Var(t)$ denotes the set of variables occurring in term t. If $Var(t) = \emptyset$, then t is called *closed* or *ground*. The set of closed terms is denoted with Ter_0 or $\mathcal{T}(\Sigma)$, where $Ter_0 \subset Ter$.
$\Sigma(t)$ is used to denote the set of symbols of Σ occurring in term t.

Example 1: Consider the signature $\Sigma = \{0, succ, =\}$ with arity $0, 1, 2$ respectively. Given variables $x, y \in V$, the terms $succ(x)$ and $succ(succ(y))$ are examples of terms in the set Ter on Σ and V. The terms 0, $succ(0)$, $succ(succ(0))$ are in Ter_0. •

Remark 1: The set V and the relation $=$ are in every first-order language, and they are also called *logical symbols*.
The terms with the equality symbol $=$, e.g. $=(x, y)$, are considered *boolean* terms. This also holds for other predicate symbols.
Note that it always holds $V \cap \Sigma = \emptyset$, even though it might not be explicitly stated in the following. ▪

Remark 2: The expression "first-order" means that the (free) variables are *individual*, that is they can be replaced by any term. From the semantic point of view, they can only be interpreted as individuals (and not, for example, as a set of individuals) in the semantic domain used to interpret the given language. Obviously, in a first-order language describing sets, variables are interpreted as sets, and sets of sets will not be denoted in such a language. ▪

In the following it will be necessary to deal with the structure of terms. Le us begin by representing terms by means of trees.

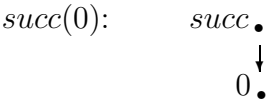
2.1 Term representation as labelled ordered trees

Each element $t \in Ter$ can be represented in a natural way as a finite labelled ordered tree, whose nodes are labelled with elements in $\Sigma(t) \cup Var(t)$. Variables can only label the tree leaves, except for the case when the term is a variable (in this case the leaf of the tree coincides with its root). Let us show some examples of terms with the corresponding trees.

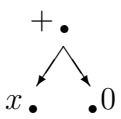
- 0:

- x:

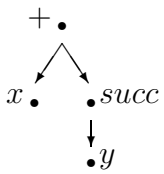
- $succ(0)$:



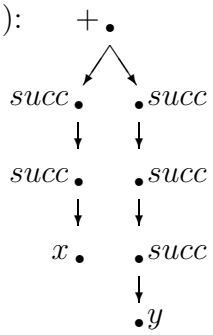
- $+(x, 0)$:



- $+(x, succ(y))$:



- $+(succ^2(x), succ^3(y))$:



Let us remark that these trees are finitely branching, because the arity of any function symbol is finite and the number of the branches exiting a node gives the arity of the function symbol labelling that node. The leaves of a tree are thus labelled with variables or constants.
Trees are ordered because the term $f(x, y)$ is different from $f(y, x)$, as equality on terms is syntactic identity.

2.2 Subterms, occurrences, positions, contexts

Subterm of a term t is a term that corresponds to a subtree of the tree T corresponding to t. For example, $succ(0)$ is a subterm of $+(succ(0), x)$. To denote that s is a subterm of t we can simply write $s \in t$. The notion of subterm is distinct from that of *occurrence*, as a subterm can have more occurrences in a term, like $succ(0)$ in $+(succ(0), succ(0))$. $Os(t)$ is the multiset of all occurrences of s in t. Whenever $Os(t) = \emptyset$, s is not a subterm of t. $O(t)$ is the multiset of all occurrences of all subterms of t. To single out a particular occurrence of a subterm, it is useful the notion of *position*, while

for denoting that a term is a subterm of another term it is sufficient to use the notion of *context*, as defined below.

Let us first introduce the notion of *position*.

Definition 2 *Given $t \in \mathcal{T}(\Sigma, V)$, the set $Pos(t)$ of the* positions *of t is a set of strings of positive integer numbers, inductively defined as follows:*

i) if $t \in V$ then $Pos(t) = \{\epsilon\}$ where ϵ denotes the empty string
ii) if $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma^n$ and $t_i \in \mathcal{T}(\Sigma, V)$, then
$Pos(t) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i.p \mid p \in Pos(t_i)\}$
where . denotes the operation of string concatenation.

Positions will be denoted with p, q, p', etc. The set of positions is partially ordered with respect to the *prefix ordering* defined as follows: for any two positions p and q, we have that $p \leq q$ if and only if there exists a position p' such that $p.p' = q$. In this case we say that p is a *prefix* of q. If two positions p and q are not comparable with respect to the prefix ordering, then p and q are said *parallel* or *disjoint*, written $p \parallel q$.

Hence, a position in a term t is a string of integers denoting the path leading to an occurrence of a subterm s in the tree T of t; more precisely, a position in a term t is a string of integers denoting the path leading to the node which is the root of the subtree corresponding to an occurrence of a subterm s of t.

Definition 3 *Given $t \in \mathcal{T}(\Sigma, V)$ and $p \in Pos(t)$, the* subterm *of t occurring at position p, witten $t|_p$, is defined inductively on the length of p:*

$$\begin{aligned} t|_\epsilon &= t \\ f(t_1, \ldots, t_n)|_{i.q} &= t_i|_q \end{aligned}$$

or, equivalently,

$$t|_p = \begin{cases} t & \text{if } p = \epsilon \\ t_i|_q & \text{if } p = i.q \text{ and } t = f(t_1, \ldots, t_i, \ldots, t_n) \end{cases}$$

Definition 4 *Given $t, t' \in \mathcal{T}(\Sigma, V)$ and $p \in Pos(t)$, the* term obtained from t by replacing the subterm at position p with t', *written $t[t']_p$, is defined by induction on the length of p:*

$$\begin{aligned} t[t']_\epsilon &= t' \\ f(t_1, \ldots, t_n)[t']_{i.q} &= f(t_1, \ldots, t_i[t']_q, \ldots, t_n) \end{aligned}$$

Indeed, if $p = i.q$ the term t has the form $f(t_1, \ldots, t_n)$ for some function symbol $f \in \Sigma^n$ and terms $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, V)$, $1 \leq i \leq n$.
As already mentioned, p is sometimes used as $t|_p \in O(t)$.

Sometimes, it is necessary to distinguish between variable positions and non-variable positions. The set of *non-variable positions* of a term t is

$$Pos'(t) = \{\ p \in Pos(t) \mid t|_p \notin V\ \}.$$

Example 2: Let $a, f, g \in \Sigma$ and $x, y \in V$. Given the term $t = f(x, g(y, a, a))$, its set of position is $Pos(t) = \{\epsilon, 1, 2, 2.1, 2.2, 2.3\}$ and the subterms are: $t|_\epsilon = f(x, g(y, a, a))$, $t|_1 = x$, $t|_2 = g(y, a, a)$, $t|_{2.1} = y$, $t|_{2.2} = a$, $t|_{2.3} = a$. The two occurrences of the constant a are thus identified. The set of non-variable positions is $Pos'(t) = \{\epsilon, 2, 2.2, 2.3\}$. •

Whenever it is not necessary to give the position of an occurrence, because we simply want to say that a term s is a subterm of a term t, it is enough to use the notation of context and write $t = C[s]$, where $C[\,]$ is called a *context* and is obtained from a term by deleting an occurrence of a subterm. The symbol $[\,]$ stands for the empty occurrence.
For example, in $f(g(a), a) = C[a]$, the context $C[\,]$ is either $f(g(a), [\,])$ or $f(g([\,]), a)$.
We can also use $C[a, a]$ to stress both occurrences of a by means of the 'double' context $C[\ ,\ ] = f(g([\,]), [\,])$. In general, a context with n empty occurrences can be used for denoting n occurrences of a subterm in a given term (without giving their positions).
Frequently, we also have the notation $t[a]$ in place of $t = C[a]$.
If, whenever using the context notation, we also want to stress the position, a redundant notation $C[\,]|_p$ can be used, so that $C[s]|_p$ denotes the subterm s at position p.

2.3 Substitutions

The typical feature of variables is that they can be replaced by means of substitutions.

Definition 5 *Let Σ be a signature and V be a countable set of variables. A* substitution *is a function $\sigma : V \to \mathcal{T}(\Sigma, V)$ such that $\sigma(x) \neq x$ only for a*

finite number of variables. For any substitution σ we define:
i) the domain $\mathcal{D}(\sigma)$*, as the (finite) set of variables replaced by σ:*

$$\mathcal{D}(\sigma) = \{x \in V \mid \sigma(x) \neq x\} \ , \ \ \sigma(x) \in \mathcal{T}(\Sigma, V)$$

ii) the set $\mathcal{I}(\sigma)$ of the variables introduced by σ*:*

$$\mathcal{I}(\sigma) = \bigcup_{x \in \mathcal{D}(\sigma)} Var(\sigma(x)).$$

Thus, a substitution can be represented as a finite set of bindings of the form $\{\sigma(x)/x \mid x \in \mathcal{D}(\sigma)\}$. Each substitution $\sigma : V \rightarrow \mathcal{T}(\Sigma, V)$ can be extended to a function $\sigma' : \mathcal{T}(\Sigma, V) \rightarrow \mathcal{T}(\Sigma, V)$ as follows: for $x \in V$ we have $\sigma'(x) = \sigma(x)$ and for a non-variable term $t = f(t_1, \ldots, t_n)$ we have $\sigma'(t) = f(\sigma'(t_1), \ldots, \sigma'(t_n))$.

A substitution σ is said *closed* if and only if $Var(\sigma(x)) = \emptyset$ for every $x \in \mathcal{D}(\sigma)$. *id* denotes the identity or empty substitution, i.e. $\mathcal{D}(id) = \emptyset$. The composition of two substitutions σ and θ is the function $\theta \circ \sigma$ defined as $(\theta \circ \sigma)(x) = \theta(\sigma(x))$ for every x, which is still a substitution: the set of substitutions is closed under composition.

Sometimes, as it will be shown in the following, substitutions will be required to be *acyclic*, that is $x \notin Var(\sigma(x))$ for all $x \in \mathcal{D}(\sigma)$. A stronger condition is that of *idempotency*.

Definition 6 *A substitution σ is* idempotent *if no element in $\mathcal{D}(\sigma)$ is a subterm of any element of the codomain of σ.*

Idempotency of substitutions means that by applying an idempotent substitution to a term in Ter, we always obtain a term in Ter, that is the term does not become an infinite term.
Idempotent substitutions are not closed under composition.

2.4 Matching substitutions and unifiers

Definition 7 *Let s and t be terms in $\mathcal{T}(\Sigma, V)$. We say that s* subsumes *t, written $s \leq t$, if and only if there exists a substitution σ such that $\sigma(s) = t$. If such σ exists, σ is called a* matching substitution *or simply a* match *of s and t. The term t is an* instance *of term s.*

The partial ordering on terms can be extended to substitutions.

Definition 8 *Given two substitutions σ, σ', $\sigma \leq \sigma'$ if and only if for all $x \in V$ we have $\sigma(x) \leq \sigma'(x)$.*

Here we consider idempotent substitutions.

Definition 9 *Given $s, t \in \mathcal{T}(\Sigma, V)$, s and t are* unifiable *if and only if there exists a substitution σ such that $\sigma(s) = \sigma(t)$. If such σ exists, then σ is called a* unifying substitution *or simply a* unifier *of s and t.*

The set of all unifiers of two terms s and t is denoted $\mathcal{U}_\emptyset(s, t)$.
In general, there can be several unifiers of two terms, or no unifiers.

Definition 10 *Let $s, t \in \mathcal{T}(\Sigma, V)$. If s and t are unifiable, then there exists a most general unifier (mgu, for short) of s and t. In other words, there exists $\sigma \in \mathcal{U}_\emptyset(s, t)$ such that for all $\theta \in \mathcal{U}_\emptyset(s, t)$ we have $\sigma \leq \theta$.*

The mgu is *unique modulo variable renaming* and can be computed with the algorithm described in Section 2.5.2. Moreover, from the above definitions it follows that, when considering acyclic substitutions, the matching of terms can be seen as unification in only one direction. A match is a substitution that, when applied to *only one* of the two terms, makes them equal, while a unifier is a substitution that, when applied to *both* terms, makes them equal. In the case of generic (possibly cyclic) substitutions, this is not true in general.

Example 3: Let $a, b, f, g \in \Sigma$ and $x, y \in V$. The terms $s = f(x, a)$ and $t = f(b, y)$ are unifiable with mgu $\sigma = \{b/x, a/y\}$, but they cannot be made equal by means of a match, that is neither term subsumes the other. The terms $s' = x$ and $t' = g(x)$ are not unifiable, but they can be made equal by means of the (cyclic) match $\sigma' = \{g(x)/x\}$ applied to s', as $\sigma'(s') = t'$. •

2.5 Algorithms of symbolic manipulation on terms

2.5.1 Instantiation

The problem of instantiation can be seen as the problem of finding a solution for the disequality $s \leq t$. In the case of $s, t \in \mathcal{T}(\Sigma, V)$, the solution is a matching substitution that, when applied to s, produces the *instantiated* term t, because $\sigma(s) = t$. When such σ exists, it is unique. If t is a closed term, then the substitution σ is also closed.

In $s \leq t$, t is said an *instance* of s and s is said an *anti-instance* of t, as one can obtain s from t by replacing an occurrence of a non-variable subterm of t with a variable. For any term t, we can define the following two subsets in $(Ter, \leq)$:
- the set *Inst(t)* of all instances of t, which can be finite, infinite or empty, and
- the set *Antinst(t)* of all anti-instances of t, which is always finite and non-empty. In fact, a variable is an anti-instance of any term.

2.5.2 Unification

The problem of unification is that of finding a solution for the equation $s = t$. In the case of $s, t \in \mathcal{T}(\Sigma, V)$, the solution is a substitution that, when applied to both terms s and t, produces the same *unified* term. In this case we talk of *syntactic unification*, opposed to *semantic unification* that will be treated at the end of these notes.

The unification problem is in general undecidable for any language. In particular, this problem becomes decidable for first-order languages, such as $\mathcal{T}(\Sigma, V)$, and various algorithms have been given which differ for their efficiency. Here, we consider the *Algorithm of the Solved Form*, which is also known as the *Algorithm by Martelli-Montanari*.

Definition 11 *A system of equations is* in solved form *if it has the following form:*

$$\begin{aligned} x_1 &= t_1 \\ &\dots \\ x_n &= t_n \end{aligned}$$

where $i \neq j$ implies $x_i \neq x_j$, $i, j = 1, \dots, n$, and every x_i has a unique occurrence in the system (that is, the one on the left-hand side of the equation).

Note that a system of equations in solved form is a way to represent an idempotent unifier $\{t_1/x_1, \dots, t_n/x_n\}$ that, once applied to the two terms to be unified, yields the unified term. The idea of the algorithm is thus to find the solution (a substitution) of a system of equations E, if it exists, by trying to transform such a system into a system in solved form, E_{ris}, which is

empty at the beginning of the procedure. This is done by applying in a non-deterministic manner the inference rules given in Fig. 1, so that the following derivation is generated in case of successful termination of the algorithm

$$(E, \emptyset), \ldots, (E', E'_{ris}), \ldots, (\emptyset, E_{ris}).$$

(**Delete**)	$\dfrac{(E \cup \{t = t\}, E_{ris})}{(E, E_{ris})}$
(**Decomposition**)	$\dfrac{(E \cup \{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\}, E_{ris})}{(E \cup \{t_1 = s_1, \ldots, t_n = s_n\}, E_{ris})}$
(**Failure 1**)	$\dfrac{(E \cup \{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\}, E_{ris}), f \neq g}{\textit{failure and termination}}$
(**Failure 2**)	$\dfrac{(E \cup \{x = t[x]\}, E_{ris}), \quad x \in Var(t)}{\textit{failure and termination}}$
(**Var. Elimination**)	$\dfrac{(E \cup \{x = t\}, E_{ris}), \quad x \notin Var(t)}{(\{t/x\}E, E_{ris} \cup \{x = t\})}$

Figure 1: Rules for the solved form.

Remark 3: The substitution $\theta = E_{ris}$ that is computed by the algorithm on its successful termination, i.e. whenever E becomes $\emptyset$ without failure, is idempotent, that is $\theta(\theta(x)) = \theta(x)$ for all x. This derives from the fact that each variable on the left-hand side of each equation has unique occurrence in the system in solved form. ∎

Proposition 1 1. *The algorithm of the solved form terminates.*
2. *The algorithm computes a system of equations in solved form or (aut) fails.*

Proposition 2 *The algorithm of the solved form is correct.*

Hence, when the algorithm computes a unifier for two terms, such terms are unifiable and the unified term belongs to $Inst(s) \cap Inst(t)$. However, does the algorithm capture all unifiable cases?

Proposition 3 *The algorithm of the solved form is complete.*

Proposition 4 *The substitution computed by the algorithm of the solved form is a most general unifier.*

2.5.3 Anti-unification

We have seen that unifying two terms consists of finding their more general common instance, when such an instance exists. Anti-unifying two terms consists of finding their more particular common anti-instance, that always exists. The problem of anti-unification is decidable.

A simple algorithm for building the anti-unified term of two terms in Ter consists of examining the two terms in parallel from left to right and, while keeping the corresponding symbols equal, we:
- maintain a variable if one of the two corresponding symbols is a variable,
- introduce a new variable if the corresponding symbols are both constant (paying attention to giving the same names to the new variables that must have the same values).

Example 4: Let $a, b, c, f, g, h \in \Sigma$ and $x, y, z \in V$. The anti-unified term of the terms $f(a, b)$ and $g(a, b)$ is x, the anti-unified term of $f(a, x)$ and $f(y, b)$ is $f(y, x)$, and the anti-unified term of $f(x, h(y, a))$ and $f(b, h(c, x))$ is $f(x, h(y, z))$. •

3 Equational calculus

Definition 12 *Let Σ be a signature and V be a set of variables. A* Σ-identity *(or* Σ-equation*) is a pair $(l, r) \in \mathcal{T}(\Sigma, V) \times \mathcal{T}(\Sigma, V)$, written $l = r$. The terms l and r are said* left-hand side *and* right-hand side*, respectively, of the identity $l = r$.*

Note that many authors do not make distinction between identities and equations, and it is often said that a set of identities is (the base of) an *equational theory*. In fact, there is a logical difference between identities and equations. Identities are equalities whose variables are implicitly universally quantified, that is equalities that are assumed to be true for any values assigned to variables. Equations are instead equalities that typically we want to *solve*, whose variables are implicitly existentially quantified: thus, we want to find those values that, once replaced to the variables, make the equation true. In the following we will indifferently talk of identities and equations, while keeping the logical distinction between the two notions.

3.1 Equational deduction

Identities can be used for transforming terms into other "equivalent" terms by replacing instances of left-hand sides with the corresponding instances of right-hand sides and vice versa.

Definition 13 *Let E be a set of Σ-identities. The* reduction relation $\longrightarrow_E \subseteq \mathcal{T}(\Sigma, V) \times \mathcal{T}(\Sigma, V)$ *is defined as follows: given $s, t \in \mathcal{T}(\Sigma, V)$,*

$s \longrightarrow_E t$ *if and only if* $\exists\, (l = r) \in E$ $\exists p \in Pos(s)$ $\exists \sigma$ *substitution such that* $s|_p = \sigma(l)$ *and* $t = s[\sigma(r)]_p$.

Example 5: Let $\Sigma = \{e, i, f\}$ be a signature where e is a constant symbol, i is a unary function symbol and f is a binary function symbol. Let $x, y, z \in V$ and E be the set with the following identities:

$$\begin{aligned} f(x, f(y, z)) &= f(f(x, y), z) && (1)\\ f(e, x) &= x && (2)\\ f(i(x), x) &= e && (3) \end{aligned}$$

The identities in E provide an equational presentation of the group theory and can be used for reducing terms to their normal forms with respect to such a theory. Let $t = f(i(e), f(e, e))$. This term can be reduced to the normal form e by means of two different sequences of reduction steps. In the first one the identities (1), (3) and (2) are applied:

$$f(i(e), f(e, e)) \longrightarrow_E f(f(i(e), e), e) \longrightarrow_E f(e, e) \longrightarrow_E e$$

while (2) and (3) are used in the second sequence:

$$f(i(e), f(e, e)) \longrightarrow_E f(i(e), e) \longrightarrow_E e.$$

•

The convertibility relation $\overset{*}{\longleftrightarrow}_E$ is the reflexive-transitive-symmetric closure of $\longrightarrow_E$ [12]. When doing equational reasoning, we would like to decide whether two terms are *equal modulo a theory* which is defined through a set E of equations, that is $s \overset{*}{\longleftrightarrow}_E t$ for $s, t \in \mathcal{T}(\Sigma, V)$. The relation $\overset{*}{\longleftrightarrow}_E$ can be syntactically characterized as follows.

Definition 14 *Let $\mathcal{R}$ be a binary relation over $\mathcal{T}(\Sigma, V)$.*
i) $\mathcal{R}$ is closed with respect to substitutions *if and only if $s\,\mathcal{R}\,t$ implies $\sigma(s)\,\mathcal{R}\,\sigma(t)$ for all $s, t \in \mathcal{T}(\Sigma, V)$ and substitution σ.*
ii) $\mathcal{R}$ is closed with respect to the operations in Σ *if and only if $s_1\,\mathcal{R}\,t_1,\ \ldots,\ s_n\,\mathcal{R}\,t_n$ implies $f(s_1, \ldots, s_n)\,\mathcal{R}\,f(t_1, \ldots, t_n)$ for all $n \geq 0$, $f \in \Sigma^n$ and $s_1, \ldots, s_n, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, V)$.*

Proposition 5 *Let E be a set of Σ-identities. The relation $\overset{*}{\longleftrightarrow}_E$ is the smallest equivalence relation over $\mathcal{T}(\Sigma, V)$ that contains E and is closed with respect to substitutions and the operations in Σ.*

Hence, starting from the set E of Σ-identities, the relation $\overset{*}{\longleftrightarrow}_E$ can be obtained by closing E with respect to reflexivity, symmetry, transitivity, substitutions and operations in Σ. This procedure of closure can be described through a system of *inference rules* that allow one to carry out *equational deduction*, i.e. derive from E all equations $s = t$ that are *syntactic consequences of E*, written $E \vdash s = t$. The following is an alternative characterization of the relation $\overset{*}{\longleftrightarrow}_E$.

Definition 15 *Let E be a set of Σ-identities. The* syntactic consequences of E *are derived through the following inference rules:*

$$\frac{(s = t) \in E}{E \vdash s = t}$$

$$\frac{}{E \vdash t = t} \qquad \frac{E \vdash s = t}{E \vdash t = s} \qquad \frac{E \vdash s = t \quad E \vdash t = u}{E \vdash s = u}$$

$$\frac{E \vdash s = t}{E \vdash \sigma(s) = \sigma(t)} \qquad \frac{E \vdash s_1 = t_1 \quad \ldots \quad E \vdash s_n = t_n}{E \vdash f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)}$$

Proposition 6 *Let E be a set of Σ-identities. The relation $\overset{*}{\longleftrightarrow}_E$ generated from E is the set of identities that are obtained by applying the inference rules of the deductive system given in Definition 15.*

Thus, for all $s, t \in \mathcal{T}(\Sigma, V)$, we have $s \overset{*}{\longleftrightarrow}_E t$ if and only if $E \vdash s = t$ in a finite number of steps using the rules of the deductive system. Such rules allow one to produce *proof trees* [1].

3.2 Models

So far we have introduced notions at syntactic level without providing a semantic interpretation of the symbols under consideration and have transformed terms by manipulating their symbolic representations. Given a signature Σ, a Σ-*model* gives an interpretation of the function symbols in Σ. For definitions on models, validity, satisfiability, variety and semantic consequences of a set of equations, please refer to [10] (in Italian, Section 3.2). Here, we simply recall the semantic definition of an equational theory as follows. Let $E \models s = t$ denote that the identity $s = t$ is a *semantic consequence* of E. The relation $=_E = \{(s,t) \in \mathcal{T}(\Sigma, V) \times \mathcal{T}(\Sigma, V) \mid E \models s = t\}$ is the *equational theory* induced by E, where E is the *base* or the *set of axioms* of the theory.

In what follows, when considering an equational theory, it will be sufficient to give the axioms E of the theory and we will talk of the equational theory E with some abuse of notation.

The relation $\overset{*}{\longleftrightarrow}_E$ is the smallest equivalence relation over $\mathcal{T}(\Sigma, V)$ that is closed with respect to operations in Σ and substitutions. Starting from E and by applying the inference rules of the deductive system given in Definition 15, all equations that are *true* in the theory can be derived. This is what is asserted by the well-known **Birkhoff's completeness theorem**, i.e. the semantically defined (*model-theoretic*) relation $=_E$ coincides with the syntactically defined (*proof-theoretic*) relation $\overset{*}{\longleftrightarrow}_E$.

Theorem 1 *Let E be a set of equations and $s = t$ be an equation. Then we have that $s =_E t$ if and only if $E \vdash s = t$.*

4 Term rewriting systems

A term rewriting system is a particular abstract reduction system $\langle A, \longrightarrow \rangle$, where A is a set of terms and $\longrightarrow$ is a reduction relation, also called *rewriting relation*. Various well-known algorithmic systems (Turing machines, λ-calculus, etc.) can be formalized as term rewriting systems. Here we consider first-order term rewriting systems, where the set of terms is $\mathcal{T}(\Sigma, V)$ and the rewriting relation is defined through a set of rewriting rules over $\mathcal{T}(\Sigma, V)$.

Let us recall that the semantics of a functional programming language, such as ML, SML, CAML, etc. is essentially given through a rewriting system. The definition of a function in a functional language is that of a rewrit-

ing system, i.e. a set of rules that assert how to evaluate the application of such function to given arguments. The evaluation of an expression in this language is a process of transformation (rewriting) of the given expression until it is not possible any more to apply further transformations. The result of this process is the value of the expression, i.e. its normal form with respect to the transformation rules.

The process of transformation to normal form can be performed by applying different evaluation (or reduction) strategies. A strategy defines how to choose, at each step, the subexpression to reduce first among all reducible subexpressions (redexes). We have, for example, the evaluation strategy "call-by-value" (like in the languages of the ML family) and the evaluation strategy "lazy evaluation" (like in the languages Miranda and Haskell).

Definition 16 *A* term rewriting system *(*trs *for short) or* rewrite system R *over a signature* Σ *is a set* $\{(l_i, r_i) \mid l_i, r_i \in \mathcal{T}(\Sigma, V), l_i \notin V, Var(r_i) \subseteq Var(l_i)\}$. *The pairs* (l_i, r_i) *are called* rewrite rules *and are written* $l_i \longrightarrow r_i$ *(oriented equations).*
The rewrite relation $\longrightarrow_R$ *over* $\mathcal{T}(\Sigma, V)$ *is defined as the smallest relation that contains* R *and is closed with respect to substitutions and the operations in* Σ *(or contexts):*

i) $t \longrightarrow_R s$ *implies* $\sigma(t) \longrightarrow_R \sigma(s)$ *for all substitution* σ;
ii) $t \longrightarrow_R s$ *implies* $f(\ldots, t, \ldots) \longrightarrow_R f(\ldots, s, \ldots)$ *for all* $f \in \Sigma^n,\ n > 0$.

4.1 Redex, derivation, normal form

Definition 17 *Given a trs* R *over* Σ, *a term* t rewrites *(or* reduces*) to a term* s, *written* $t \longrightarrow_R s$, *is there exist a rule* $l \longrightarrow r$ *in* R, *a substitution* σ *and a* redex $t|_p$ *at position* p, *such that* $t|_p = \sigma(l)$ *and* $s = t[\sigma(r)]_p$.

Example 6: Let R be the following trs:

$$\begin{aligned} f(h(x)) &\longrightarrow h(x) \\ h(h(x)) &\longrightarrow x \\ h(a) &\longrightarrow a \end{aligned}$$

Given the term $t = h(f(h(b)))$, let us check which reductions can be applied in R starting from t. The positions of t are $Pos(t) = \{\epsilon, 1, 1.1, 1.1.1\}$. The only possible reduction is with the first rule, $p = 1$, $t|_p = f(h(b))$ and $\sigma = \{b/x\}$,

and we have $t \longrightarrow_R t' = t[\sigma(h(x))]_p = h(h(b))$. Given $Pos(t') = \{\epsilon, 1, 1.1\}$, we have another reduction step with the second rule, $p = \epsilon$, $t'|_p = t'$ and $\sigma' = \{b/x\}$, thus $t' \longrightarrow_R t'' = t'[\sigma'(x)]_p = b$. The reduction sequence is as follows: $h(f(h(b))) \longrightarrow_R h(h(b)) \longrightarrow_R b$. •

From now on we will write $\longrightarrow$ instead of $\longrightarrow_R$, whenever R is known from the context. Recall that a term t is *in normal form* with respect to a trs R if and only if there is no term s such that $t \longrightarrow s$, that is no subterm of t is an instance of the left-hand side of some rewrite rule in R. Moreover, a term s is *a normal form* of a term t with respect to R if and only if $t \stackrel{*}{\longrightarrow} s$ and s is in normal form with respect to R. The following notation will sometimes be used: given two terms s, t and a relation $\longrightarrow$, $s \downarrow t$ denotes that s and t have a common successor, i.e. there exists a term u such that $s \stackrel{*}{\longrightarrow} u \stackrel{*}{\longleftarrow} t$. A normal form of a term t with respect to a trs R will be written $t\downarrow$.

5 Terminating rewrite systems

We know that the termination property (SN) of a rewrite system ensures the existence of the normal form for any term, as there are no sequences with an infinite number of reduction steps [12]. Unfortunately, the problem of the termination of a rewrite system is, in general, undecidable. This result holds even if the rewrite rules contain only unary function symbols, or if we consider a rewrite system with only one rule with function symbols whose arity is greater than 1.
In fact, every Turing machine can be formalized as a rewrite system such that, for each computation step of a Turing machine, there is a reduction in the corresponding rewrite system: thus, the decidability of termination in rewrite systems would make the *Halting Problem* decidable.
The termination problem is decidable in the case of closed rewrite systems, whose rules do not contain any variable.
The termination problem is common to various contexts, more or less abstract, in both Computer Science and Mathematics. Typically, for deciding the termination of rewrite systems, we can use:

- partial orderings on terms having useful properties for the termination problem;
- *embedding* of a system into another one that is known to be terminating.

The undecidability, in general, of termination of a (finite) rewrite system forces one to look for sufficient conditions which allow one to establish whether a rewrite system is terminating or not. The idea is based on the intuition of what a rewrite system does, that is when reducing a term by means of a rewrite relation, the structural complexity of the resulting term be, in a certain sense, smaller than that of the starting term. Obviously, this intuition does not agree with the notion of infinite rewriting and is thus connected with the notion of termination. The aim is that of establishing a partial ordering $\succ$ on the set of terms $\mathcal{T}(\Sigma, V)$, such that we can prove that, if $t \longrightarrow s$, then we have $t \succ s$. Hence, it would be sufficient to find a well-founded ordering $\succ$ on terms, such that $t \longrightarrow s$ implies $t \succ s$ for all terms t, s. This means that a rewrite system R is terminating if and only if the rewrite relation $\longrightarrow_R$ is contained in the relation $\succ$.

However, this result is not very helpful as it expresses a condition on all possible reductions. Instead, we want to give a similar result that takes into consideration the term structure and the way terms can reduce. A rewrite relation is, by definition, closed with respect to contexts and substitutions. The aim is to define orderings on terms that are well-founded and closed under contexts and substitutions, so that, once we have an ordering with such properties, in order to check whether a rewrite system is terminating, it is sufficient to verify a condition on the rules of the rewrite system (and the rules are assumed to be a finite number), opposed to a condition on all possible reductions (that can be an infinite number).

5.1 Partial orderings useful for termination

Let us consider reduction orderings and some particular cases.

5.1.1 Reduction orderings

The relation $\succ \subset \mathcal{T}(\Sigma, V) \times \mathcal{T}(\Sigma, V)$ is a *reduction ordering* if it is:

- partial;
- strict (irreflexive and transitive);
- antisymmetric;
- well-founded or noetherian;

- closed under contexts (also said *monotone* or *compatible with operations*):
$$t \succ t' \Longrightarrow C[t] \succ C[t']$$
for any context $C[]$, or equivalently
$$t \succ t' \Longrightarrow f(\ldots, t, \ldots) \succ f(\ldots, t', \ldots)$$
for any $f \in \Sigma$;
- closed under substitutions (also said *stable*):
$$t \succ t' \Longrightarrow \sigma(t) \succ \sigma(t')$$
for any substitution σ.

Example 7: Let $t \succ t'$ if $|t| > |t'|$, where $|.| : \mathcal{T}(\Sigma, V) \longrightarrow \mathbf{N}$, with $|t| \geq 1$, gives the number of characters that are in t when is considered as a string. Verify that $\succ$ is not a reduction ordering, as the closure under substitutions is not satisfied. •

Example 8: Let $t \succ t'$ if $|t| > |t'|$ and $|t|_x > |t'|_x$ for each variable x, where $|t|_x$ gives the number of occurrences of the variable x in t.
Verify that $\succ$ is a reduction ordering. •

Proposition 7 *(Lankford Theorem) A rewrite system R is terminating if and only if there exists a reduction ordering $\succ$ such that $l \succ r$ for each $l \longrightarrow r$ in R.*

Proof (*the proof has not been presented at the lectures, thus is not part of the programme of the course*)
If $l \succ r$ for each $l \longrightarrow r$ in R, then $\longrightarrow_R \subseteq \succ$ and also $\overset{*}{\longrightarrow}_R \subseteq \succ$, as $\succ$ is transitive. Moreover, as $\succ$ is well-founded, each reduction is finite, since the relation $\longrightarrow_R$ is induced by the rules.
Vice versa, if R is terminating, we can define $t \succ t'$ if $t \overset{+}{\longrightarrow}_R t'$.
Since $\longrightarrow_R \subseteq \overset{+}{\longrightarrow}_R$, we have $l \succ r$ if $l \longrightarrow r$ for every rule in R.
This ordering is a reduction ordering, because from the fact that R is terminating it follows that $\succ$ is irreflexive, transitive, antisymmetric, well-founded and closed under substitutions and contexts. ∎

Simplification orderings are particular reduction orderings.

5.1.2 Simplification orderings

The relation $\succ \subset \mathcal{T}(\Sigma, V) \times \mathcal{T}(\Sigma, V)$ is a *simplification ordering* if it is:

- partial;
- strict;
- antisymmetric;
- has the *subterm property* (property stronger than well-foundedness), that is:
$$t = C[t'] \succ t'$$
for any context $C[] \neq []$ of t'. Using positions, an equivalent definition is
$$t \succ t|_p$$
for any position $p \in Pos(t) \backslash \{\epsilon\}$;
- closed under contexts;
- closed under substitutions.

It is easy to define simplification orderings, as each proper subterm t' of t is such that $t \succ t'$.

Example 9: Verify that $>_{lex}$ is not a simplification ordering, as the subterm property is not satisfied (if $f >_{lex} g$, then $f(g(f(x))) >_{lex} g(f(x))$, but it not true that $g(f(x)) >_{lex} f(x)$). •

Every simplification ordering on a finite signature contains the *embedding* ordering (for a definition of homomorphic embedding, please refer to [1]).

Theorem 2 *Let Σ be a finite signature. Every simplification ordering $\succ$ on $\mathcal{T}(\Sigma, V)$ is a reduction ordering.*

For the proof we refer to [1], where the embedding ordering is used.

Recursive path orderings (*rpo* for short) are another class of particular reduction orderings.

5.1.3 Recursive path orderings

Before defining recursive path orderings, we must extend orderings on elements to finite multisets of elements.
Let $(A, \succ)$ be a set A (in our setting $A = \mathcal{T}(\Sigma, V)$) equipped with a partial ordering $\succ$. The ordering $\succ$ is extended to the ordering $\succ\!\!\succ$ on finite multisets on A, thus yielding $(MA, \succ\!\!\succ)$, with the following definition (with M, M_1, M_2 meta-variables for multisets).

Definition 18 *$M_1 \succ\!\!\succ M_2$ if $M_1 \neq M_2$ and for all $m_2 \in M_2 \backslash M_1$ there exists $m_1 \in M_1 \backslash M_2$ such that $m_1 \succ m_2$.*

Example 10: $M_1 = \{3, 3, 4, 0\} \succ\!\!\succ M_2 = \{3, 2, 2, 1, 1, 1, 4\}$ as
$M_2 \backslash M_1 = \{2, 2, 1, 1, 1\}$
$M_1 \backslash M_2 = \{3, 0\}$
and we have $3 \succ 2$ and $3 \succ 1$.
Instead, the relation does not hold between $M_1 = \{3, 1, 1\}$ and $M_2 = \{4, 1\}$ as $M_2 \backslash M_1 = \{4\}$, $M_1 \backslash M_2 = \{3, 1\}$ and we have that neither $3 \succ 4$ nor $1 \succ 4$.
•

Proposition 8 *If $\succ$ is well-founded, then $\succ\!\!\succ$ is also well-founded.*

The ordering $\succ\!\!\succ$ can also be defined as the transitive closure of the replacement of an arbitrary element in a multiset with a finite number (possibly zero) of elements, which are smaller in $\succ$ than the given element.

We now define the ordering *rpo* for closed or ground terms, and then extend rpo to terms with variables. The rpo is parametric on an ordering $>$, defined on the signature Σ, that establishes the *precedences* (or *priorities*) of the operators.

Definition 19 *Let $(\Sigma, >)$ be the signature equipped with a partial ordering $>$ on Σ. The ordering* rpo *on $\mathcal{T}(\Sigma)$ is recursively defined as follows:*

$$s = f(s_1, \ldots, s_m) \succ_{rpo} t = g(t_1, \ldots, t_n)$$

if:

i) $f = g$ and $\{s_1, \ldots, s_m\} \succ\!\!\succ_{rpo} \{t_1, \ldots, t_n\}$

ii) $f > g$ and $\{s\} \succ\!\!\succ_{rpo} \{t_1, \ldots, t_n\}$

iii) $f \not\geq g$ *and* $\{s_1, \ldots, s_m\} \succcurlyeq\!\!\!\succ_{rpo} \{t\}$

where $\succcurlyeq\!\!\!\succ_{rpo}$ *is* $\succ\!\!\!\succ_{rpo}$ *or the equality* $\sim$*, by taking into account that two terms are considered equal if they are equal modulo permutation of the arguments.*

Thus, the two terms $f(t_1, t_2) = f(t_2, t_1)$ are considered equal according to the above definition, as they are equal modulo permutations of the arguments.

The ordering *rpo* is extended to terms with variables by simply adding to the clauses in Definition 19 a further clause that deals with the introduction of variables, so that the resulting ordering on terms is stable.

Definition 20 *The* generalized rpo *on* $\mathcal{T}(\Sigma, V)$ *is defined as follows: given two terms* s *and* t*, we have that* $s \succ_{rpo} t$ *if i), ii) or iii) of Definition 19 holds or*
iv) $s \notin V$ *and* $t \in Var(s)$*.*

Theorem 3 *Every rpo is a simplification ordering.*

Proof The reader may refer to [1]. ∎

Example 11: Let R be the following rewrite system on the signature $\Sigma = \{a, b, c, f, g, h\}$:

$$\begin{aligned} f(a,b) &\longrightarrow g(c) \\ h(f(a,c)) &\longrightarrow h(b) \\ c &\longrightarrow b \end{aligned}$$

If we take an *rpo* based on the precedences $h > g > f > a > b > c$, we can check that in $f(a,b) \longrightarrow g(c)$ we have that $f(a,b) \succ_{rpo} g(c)$ does not hold, and in $c \longrightarrow b$ it is not true that $c \succ_{rpo} b$. •

Example 12: Let R be the following rewrite system on the signature $\Sigma = \{a, f, g, h\}$:

$$\begin{aligned} f(x,a) &\longrightarrow g(x) \\ f(x,g(y)) &\longrightarrow g(f(x,y)) \\ h(x,a) &\longrightarrow x \\ h(x,g(y)) &\longrightarrow f(h(x,y),x) \end{aligned}$$

One has to give a reduction ordering on terms such that R is terminating with respect to such an ordering. It is sufficient to consider an *rpo* based on the precedences $h > f > g$. In fact, we can verify that for each rule $l \longrightarrow r$ in R we have $l \succ_{rpo} r$. •

6 Confluent rewrite systems

From the Newman Lemma on abstract reduction systems [12], we know that $SN + WCR \Longrightarrow CR$. We also know that the convertibility relation is, in general, undecidable. However, it is decidable for terminating rewrite systems. In fact, the following decidability result follows from termination and the Church-Rosser property:

Lemma 1 *If the relation $\longrightarrow$ is noetherian and Church-Rosser, then $\overset{*}{\longleftrightarrow}$ is decidable.*

Proof We have $s \overset{*}{\longleftrightarrow} t$ if and only if $s\downarrow \equiv t\downarrow$, as the normal form for any term exists and is unique in canonical rewrite systems. ∎

Thus, given a terminating rewrite system, local confluence ensures confluence. The term structure allows us to obtain a sufficient condition for local confluence. In order to express this condition, we need to introduce the notion of critical pair.

6.1 Spectrum of the rules of a rewrite system

The situations in reduction sequences that can compromise local confluence are those where a term can rewrite in two different ways. These situations can be found by considering those cases where more than one rule can be applied to the same term. As a reduction sequence is obtained by closing the rewrite rules under substitutions and contexts, we can consider the set of all possible instances of the left-hand sides of the rules in R, thus defining the notion of *spectrum* of the rules [2].

Definition 21 *Let R be a rewrite system. The* spectrum *of the rules of R is defined as follows:*

$$spectrum(R) = \{\sigma s \mid \exists t.\ (s,t) \in R,\ \sigma\ substitution\}$$

Let us consider the following set B:

$$B = \{(s,t) \mid \exists u, s', t'.\ u \in spectrum(R), s' \longleftarrow u \longrightarrow t',\ s = s'\downarrow,\ t = t'\downarrow\}.$$

Note that the definition given for B is not general, as the case of rule application to subterms is not considered. This simplification is done to make the idea of the critical situations to be considered more intuitive and immediate.

If we represent graphically the situations described by the definition of B, looking at the spectra of the rewrite rules means to examine the following rules:

$$\begin{array}{rcl} l & \longrightarrow & r \\ \sigma_1 l & \longrightarrow & \sigma_1 r \\ \sigma_2 l & \longrightarrow & \sigma_2 r \\ & \ldots & \end{array}$$

and to find those terms u that appear as equal left-hand sides $\sigma_i l = \tau_j l'$ in two different spectra:

$$\begin{array}{lll} l \longrightarrow r & & l' \longrightarrow r' \\ \sigma_1 l \longrightarrow \sigma_1 r & & \tau_1 l' \longrightarrow \tau_1 r' \\ \sigma_2 l \longrightarrow \sigma_2 r & & \tau_2 l' \longrightarrow \tau_2 r' \\ \ldots & & \ldots \\ & \sigma_i l \longrightarrow \sigma_i r & \\ & \tau_j l' \longrightarrow \tau_j r' & \\ & \ldots & \\ \ldots & & \ldots \end{array}$$

If we extend the notion to the most general situation, we need to deal with all those cases where the instances of two left-hand sides of rules are such that a subterm of one instance coincides with the whole other instance. This means that the first instance is a term that, by definition, rewrites in two different ways, because a rule is applied on the whole term and another rule is applied on the subterm. These situations are captured by the notion of critical pairs.

6.2 Critical pairs

Let us start with some examples of critical pairs derived from rewrite rules.

Example 13:

$$\begin{aligned} 0 + succ(x) &\longrightarrow succ(x) \\ y + succ(z) &\longrightarrow succ(y + z) \end{aligned}$$

Using the mgu $\sigma = \{0/y, x/z\}$, we have $\sigma(0 + succ(x)) = \sigma(y + succ(z)) = 0 + succ(x)$. By applying the first rule to the unified term we obtain $succ(x)$, while by applying the second rule we obtain $succ(0 + x)$. These terms are both in normal form with respect to the given rules. In this way we have computed the critical pair $(succ(x), succ(0 + x))$. •

Example 14:

$$\begin{aligned} or(x, y) &\longrightarrow x \\ or(x, y) &\longrightarrow y \end{aligned}$$

With the empty substitution id we have that the term $or(x, y)$ reduces to x using the first rule, and to y using the second rule. Both terms are in normal form with respect to the given rules. The critical pair is (x, y). •

Example 15:

$$\begin{aligned} f(g(x)) &\longrightarrow b \\ g(h(a)) &\longrightarrow c \end{aligned}$$

With the mgu $\theta = \{h(a)/x\}$, from the unified term $f(g(h(a)))$ we obtain the critical pair $(b, f(c))$. •

Let us show an example of critical pair generated from only one rule, whose left-hand side unifies with (also said *overlaps on*) the left-hand side of a renamed variant of the same rule.

Example 16:

$$\begin{aligned} f(f(x)) &\longrightarrow r(x) \\ f(f(y)) &\longrightarrow r(y) \end{aligned}$$

Using the mgu $\theta = \{f(x)/y\}$, the term $f(f(x))$ unifies with $f(y)$, thus getting the unified term $f(f(f(x)))$, from which the critical pair $(r(f(x)), f(r(x)))$ is obtained. The term $r(f(x))$ derives from the redex $f(f(f(x)))$ by replacing the variable y of the second rule with $f(x)$, while the term $f(r(x))$ derives from the innermost redex $f(f(x))$ of $f(f(f(x)))$ using the first rule. •

Definition 22 (s,t) *is a* critical pair *for a rewrite system* R *if there exist two rules (with no variables in common)*

$$\begin{aligned} l_1 &\longrightarrow r_1 \\ l_2 &\longrightarrow r_2 \end{aligned}$$

a position $p \in Pos'(l_1)$ *and an mgu* θ *such that*

$$\theta(l_1|_p) = \theta(l_2)$$

so that $s = \theta(l_1)[\theta(r_2)]_p$ *and* $t = \theta(r_1)$.

Graphically we have:

$$\begin{array}{ccc} & \theta(l_1) & \\ & \swarrow \quad \searrow & \\ \theta(l_1)[\theta(r_2)]_p & & \theta(r_1) \end{array}$$

The set CC of critical pairs of a rewrite system is given by all critical pairs that can be obtained from its rules and their renamed variants. As we have seen in Example 16, after variable renaming, a rule can also overlap on itself *on a proper subterm* and generate a critical pair.

Definition 23 *A trs* R *is said* ambiguous *if its set CC of critical pairs is non-empty.*

By definition, a critical pair (s,t) of a rewrite system R contains two terms that are convertible in R, but the pair might not be convergent.

Definition 24 *A critical pair* (s,t) *of a trs* R *is* convergent *if* $s \downarrow t$.

The importance of the notion of critical pair lies in the intuitive fact that whenever a term t reduces in one step in R to two syntactically different terms t_1 and t_2, we have that t_1 has a subterm that is an instance of one element of a critical pair of R and t_2 has a subterm that is an instance of the other element of the same critical pair. This is what is asserted by the so-called **Critical Pairs Lemma**, whose proof provides the details of all syntactically possible cases (the proof can be found in [1]).

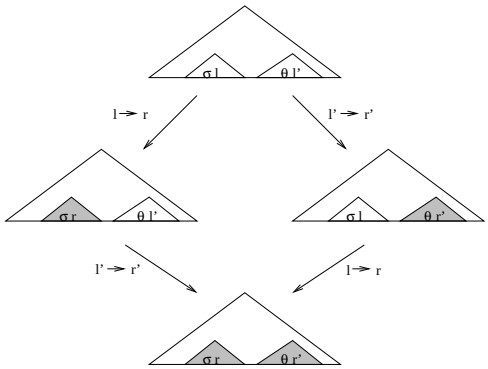
6.3 The Huet Lemma

If a rewrite system R has a finite number of rules, then its set CC of critical pairs is also finite. The following lemma provides a sufficient condition for checking local confluence.

Lemma 2 *(Huet Lemma) A rewrite system R is locally confluent (WCR) if and only if all its critical pairs are convergent.*

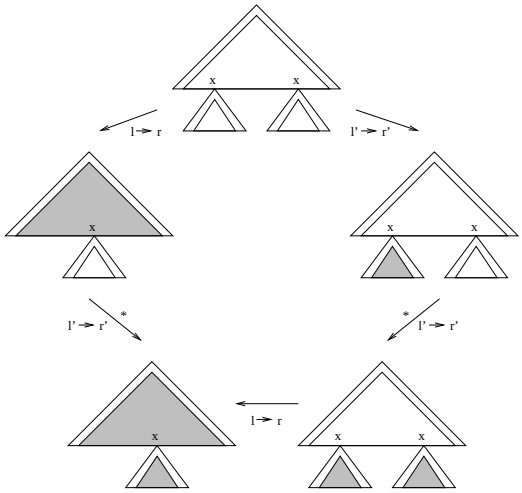
Proof (*the proof has not been presented at the lectures, thus is not part of the programme of the course*)
If R is WCR, all its critical pairs are obviously convergent. Hence, we only prove the 'if' implication "if all critical pairs of R are convergent, then R is locally confluent".
The following cases describe the three possible relations in which redexes can be inside a term. The graphical representation simplifies the proof development and makes it more intuitive.

1. Disjoint redexes



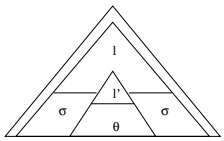
For example, the term $t = f(g(x), h(b))$ can be reduced using the rules $g(x) \longrightarrow a$ and $h(y) \longrightarrow c$, thus obtaining $f(a, h(b))$ and $f(g(x), c)$ respectively, which clearly converge to $f(a, c)$.

2. Nested redexes (variable overlapping)



For example, the term $t = h(f(g(x)), c)$ reduces using rules $g(y) \longrightarrow a$ and $f(x) \longrightarrow r(x, x)$, thus obtaining $h(f(a), c)$ and $h(r(g(x), g(x)), c)$ respectively, which clearly converge to $h(r(a, a), c)$.

3. Overlapping redexes (where the convergence hypothesis on the critical pairs is used)
The term $t = \mathcal{C}[(\sigma l)[\theta l']_p]$ can be represented as follows:



The term t reduces by the rule $l \longrightarrow r$ to $t_1 = \mathcal{C}[\sigma r]$ and by the rule $l' \longrightarrow r'$ to $t_2 = \mathcal{C}[(\sigma l)[\theta r']_p]$. By setting $\rho = \sigma \circ \theta$, we have $t_1 = \mathcal{C}[\rho r]$ and $t_2 = \mathcal{C}[(\rho l)[\rho r']_p]$. By the convergence hypothesis on the critical pairs, we have $t_1 \xrightarrow{*} \mathcal{C}[t'] \xleftarrow{*} t_2$ for some term t'.

For example, the term $t = h(f(f(f(a))))$ can be reduced in two different ways using the rule $f(f(x)) \longrightarrow b$, thus obtaining $h(b)$ and $h(f(b))$ respectively, which give a non-convergent critical pair. If the

rewrite system had further rules that make the critical pair convergent, e.g. $f(b) \longrightarrow c$ and $b \longrightarrow c$, then the rewrite system would be WCR.

■

7 Canonical rewrite systems

In this section we describe a method for deriving a *canonical* rewrite system, starting from a rewrite system that is not canonical, but admits a finite equivalent canonical rewrite system. The semantic meaning of this method, that is based on a "completion" of an equational theory, lies in the fact that it provides a decision procedure for the *word problem* in an equational theory.

7.1 The word problem

Given an equational theory E, the word problem consists of deciding whether $E \vdash s = t$, i.e. if $s = t$ modulo E. This problem can be solved by means of rewriting if there exists a rewrite system R equivalent to E, that is with the same deduction power of E. Let us recall that an equation denotes equality and thus can be applied in both directions, while a rewrite rule can be applied only in one direction. Finally, the substitutions required when rewriting are matches applied to the rewrite rules.

Definition 25 *A rewrite system R is* equivalent *to a set of equational axioms E if for all $s, t \in \mathcal{T}(\Sigma, V)$ we have*

$$s \overset{*}{\longleftrightarrow}_R t \text{ if and only if } E \vdash s = t.$$

The aim is thus to find (if it exists) a rewrite system R equivalent to E.

Theorem 4 *If R equivalent to E is canonical (or convergent or complete), then the word problem in E is decidable.*

Proof We have $s\downarrow \equiv t\downarrow$ if and only if $s \overset{*}{\longleftrightarrow}_R t$ if and only if $E \vdash s = t$. ■

Hence, we can use $\longrightarrow$ instead of $\longleftrightarrow$ without ever backtracking in reduction sequences. The vice versa of this theorem is not true. There are equational theories with a decidable word problem that do not admit a finite equivalent complete rewrite system [4].

7.2 The Knuth-Bendix Theorem

The following result, known as the **Knuth-Bendix theorem**, can be considered as a corollary of the Newman Lemma and the Huet Lemma:

Theorem 5 *A terminating rewrite system is canonical if all its critical pairs are convergent.*

Proof If all critical pairs of a rewrite system are convergent, then the system is locally confluent by the Huet Lemma. As the rewrite system is also terminating, by the Newman Lemma the system is confluent. ∎

Given a set of equational axioms specifying a particular structure, we must determine whether there exists a canonical rewrite system equivalent to the given set. Given a rewrite system R that is not complete, the problem of its completion consists in finding a set R' of rewrite rules that is equivalent and complete. Let us see how we can proceed to determine, if it exists, such canonical rewrite system.

Suppose that a terminating rewrite system is given, thus we consider the problem of confluence. From [2] we can formulate the problem as follows: given a rewrite relation $\longrightarrow_R$, determine a rewrite relation $\longrightarrow_{R'}$ such that $\overset{*}{\longleftrightarrow}_R = \overset{*}{\longleftrightarrow}_{R'}$ and $\longrightarrow_{R'}$ has the property CR.
The idea is to consider all possible situations in which confluence can be compromised, and we try to recover these cases by introducing the new pair (s, t) into the relation R', that is by adding the new rule $s \longrightarrow t$ or $t \longrightarrow s$ into the relation $\longrightarrow_{R'}$. The pair is introduced only if its insertion does not compromise the termination of the rewrite system under construction. The new rule preserves the correctness of the procedure (both s and t derive from the same term), as $\overset{*}{\longleftrightarrow}_R = \overset{*}{\longleftrightarrow}_{R'}$. If the completion procedure terminates with success (as we will see below), then R' is confluent.

If R' is terminating, we know that it is sufficient to derive the property WCR, thus it is enough to compute all its critical pairs (that are in a finite number) and then to orient them with respect to the given reduction ordering in order to preserve termination.

7.3 Completion procedures

Based on the Knuth-Bendix theorem, several completion procedures have been given in the literature, more or less efficient, that under the termination

hypothesis try to eliminate all critical pairs of a rewrite system. Here we illustrate the inference rules that must be applied non-deterministically, by choosing in an arbitrary way which rule to apply and where. Each strategy of application of the inference rules (with the possible addition of further rules) defines a specific completion procedure.

One starts with a set of equational axioms E, that has to be transformed into an equivalent canonical rewrite system, initially empty. Thus, the initial configuration is the pair $(E, \emptyset)$.

A reduction ordering $\succ$ is established to be used in some of the inference rules given in Fig. 2 in such a way that the termination property is preserved.

Remark 4: Each completion procedure explicitly depends on the chosen reduction ordering. The reduction ordering is an input parameter to the completion procedure together with the set E. The input is thus $(E, \succ)$. ∎

Orient (with similar rule if $r \succ l$):

$$\frac{(E \cup \{l = r\}, R),\ l \succ r}{(E, R \cup \{l \longrightarrow r\})}$$

Failure:

$$\frac{(E \cup \{l = r\}, R),\ l \not\succ r,\ r \not\succ l}{(failure)}$$

Delete:

$$\frac{(E \cup \{l = l\}, R)}{(E, R)}$$

Simplify on the left (with similar rule on the right):

$$\frac{(E \cup \{l = r\}, R),\ l \xrightarrow{+} l\downarrow}{(E \cup \{l\downarrow = r\}, R)}$$

Deduce:

$$\frac{(E, R),\ (c = c') \in CC}{(E \cup \{c = c'\}, R)}$$

Figure 2: Basic rules for completion.

The basic inference rules for completion are given in Fig. 2. By applying

such rules, a sequence of configurations is built

$$(E, \emptyset), (E_1, R_1), \ldots, (E_i, R_i), \ldots$$

where $(E_i, R_i) \vdash (E_{i+1}, R_{i+1})$ for each i. Such a sequence can

1. terminate with $(\emptyset, R_n)$. In this case R_n is canonical and all equations have been oriented.
   If E, besides being finite, is also closed and $\succ$ is a total ordering on closed terms, the completion procedure terminates with success [6, 14].

2. terminate with (E_i, R_i) and $E_i \neq \emptyset$, i.e. the procedure fails. This happens, for example, when a critical pair cannot be oriented with respect to the given reduction ordering. This can even happen when an equivalent complete rewrite system actually exists for E. For details on this situation we refer to [4]. Changing the reduction ordering during completion with one 'more powerful' (that preserves the ordering on those terms already compared with respect to the previous reduction ordering) may result in an incorrect completion procedure, even in the case of successful termination [13].

3. be infinite, and at the limit it can either fail or derive a canonical rewrite system with an infinite (countable) set of rewrite rules. In this case we talk of *divergence* of completion.

Besides the basic rules for completion, we can also consider the following inference rules, which perform simplifications on the rules of the current rewrite system R_i:

Compose:

$$\frac{(E, R \cup \{l \longrightarrow r\}),\ r \longrightarrow r'}{(E, R \cup \{l \longrightarrow r'\})}$$

Collapse:

$$\frac{(E, R \cup \{l \longrightarrow r\}),\ l \longrightarrow l' \text{ by } l_k \longrightarrow r_k \text{ with } l \longrightarrow r \rhd l_k \longrightarrow r_k}{(E \cup \{l' = r\}, R)}$$

where $\rhd$ is a well-founded ordering on rewrite rules defined as follows:
$l \longrightarrow r \rhd l_k \longrightarrow r_k$ if (i) a subterm of l is an instance of l_k or (ii) l and l_k are equal and $r \succ r_k$.

In a way similar to the inference rule **Simplify** that reduces to normal form the sides of the equations in E_i with respect to R_i, the inference rules **Compose** and **Collapse** allow one to reduce the rewrite rules in R_i, thus yielding an *inter-reduced* rewrite system.

7.4 Properties of the completion rules

1. The rules for completion generate a terminating rewrite system.
   In fact, each equation is transformed into a rule only if this is allowed by the given reduction ordering $\succ$.

2. The rules for completion are *correct* in the sense that if $(E_i, R_i) \vdash (E_{i+1}, R_{i+1})$, then $\overset{*}{\longleftrightarrow}_{(E_i,R_i)} = \overset{*}{\longleftrightarrow}_{(E_{i+1},R_{i+1})}$, and thus the underlying equational theory is unchanged.
   By examining each inference rule, **Orient**, **Delete** and **Deduce** are obviously correct, the remaining rules are correct based on the definition of rewrite relation.
   For the correctness of infinite completion we refer to [1], here we are mainly interested in finite rewrite systems.

Example 17: The classic example of completion by Knuth-Bendix is obtained by orienting the theory

$$\begin{aligned} 0 + x &= x \\ x + (-x) &= 0 \\ (x + y) + z &= x + (y + z) \end{aligned}$$

with respect to an rpo based on the precedence $+ > 0$ and by considering the associativity of "+" (third equation) from left to right, thus getting the incomplete system

$$\begin{aligned} 0 + x &\longrightarrow x \\ x + (-x) &\longrightarrow 0 \\ (x + y) + z &\longrightarrow x + (y + z) \end{aligned}$$

In fact, for example, from the term $0 + (-0)$ we obtain the critical pair $(-0, 0)$ which is not convergent; from $(x + (-x)) + y$ we obtain the critical pair $(y, x + (-x + y))$ which is again not convergent. •

Other examples are presented in [1]. Note that these examples should be typically given to tools implementing the completion procedure, as there might be a very high number of applications of the inference rules to carry out. Completion procedures are implemented in some automatic theorem provers, like Otter, RRL, EQP and Larch Prover [17], and in other proof assistants, like C*i*ME, ELAN and Maude [15, 16, 18].

The completion procedure has also been extended to deal with typical cases of non-termination and failure. There are extensions where critical pairs that cannot be oriented are considered as rewrite rules in both directions, and others which carry out completion *modulo* equivalence classes based on those equations of E that are kept as equations.

8 Rewriting modulo equations

When some equations of a theory E cannot be oriented, we proceed to instantiating, unifying, completing, ... modulo such equations. Rewriting modulo equations replaces the relation R with R/E everywhere and a reduction can be written as follows:

$$t_1 \longrightarrow_{R/E} t_2 =_E t_2' \longrightarrow_{R/E} \dots$$

If the equivalence classes are finite, $\longrightarrow_{R/E}$ is decidable if $\longrightarrow_R$ is decidable.

8.1 E-unification

Let us keep the set E as a set of equational axioms and formulate the unification problem as $s =_E t$, whose solutions are in Ter/E. Syntactic unification is the particular case when $E = \emptyset$. E-unification is also said *semantic unification*. The usual notions on unification are reformulated as follows, starting from the subsumption orderings on terms and substitutions.

Definition 26 *Let $s, t \in \mathcal{T}(\Sigma, V)$ and σ, δ, γ substitutions.*

- *$s \leq_E t$ if there exists σ such that $\sigma(s) =_E t$. The substitution σ is said a* matching substitution modulo E *or* E-match *of s and t.*
- *$\sigma \leq_E \delta$ if there exists γ such that $\gamma(\sigma(x)) =_E \delta(x)$ for each variable x.*
- *$\sigma =_E \delta$ if $\sigma(x) =_E \delta(x)$ for each x.*

- $\sigma \approx_E \delta$ *if* $\sigma \leq_E \delta$ *and* $\delta \leq_E \sigma$.

Note that in general $\sigma \leq_E \delta$ and $\delta \leq_E \sigma$ does not imply $\sigma =_E \delta$, as illustrated in the following example.

Example 18: Let $E = \{f(x,x) = x\} = I$ (idempotency), $\sigma = \{y/x\}$ and $\delta = \{f(y,z)/x\}$. We have that $\sigma \approx_E \delta$ because $\sigma \leq_E \delta$ with $\gamma = \{f(y,z)/y\}$ and $\delta \leq_E \sigma$ with $\gamma = \{y/z\}$, but $\sigma(x) \neq_E \delta(x)$. •

Definition 27 *Two terms* $s, t \in \mathcal{T}(\Sigma, V)$ *are* E-unifiable *if and only if there exists a substitution* σ *such that* $\sigma(s) =_E \sigma(t)$. *The substitution* σ *is called* E-unifier *of* s *and* t.

In general, there is no unique (modulo variable renaming) E-mgu, thus sets of E-unifiers must be considered as follows.

Definition 28 *Let* $s, t \in \mathcal{T}(\Sigma, V)$ *and* U *be a set of E-unifiers of* s *and* t, *that is* $U \subseteq \mathcal{U}_E(s,t)$.

- U *is minimal, written* MU, *if for all* $\sigma, \sigma' \in U$ $\sigma \leq_E \sigma' \Longrightarrow \sigma =_E \sigma'$.
- U *is complete, written* CU, *if for every E-unifier* δ *of* s *and* t *there exists* $\sigma \in U$ *such that* $\sigma \leq_E \delta$.
- U *is* CMU *if it is minimal and complete.*

All idempotent semigroups do not have complete sets of E-unifiers.

Example 19:

- Let $E = \{x + y = y + x\} = C$ (commutativity of $+$), $s = 0 + x$ and $t = y + z$. We have $0 + x =_E y + z$ through $\theta_1 = \{0/y, x/z\}$. Given $x + 0 =_E y + z$, also $\theta_2 = \{x/y, 0/z\}$ is an E-unifier of s and t, where $\theta_1 \not\leq_E \theta_2$ and $\theta_2 \not\leq_E \theta_1$. Moreover, for any other E-unifier δ (e.g. $\delta = \{0/x, 0/y, 0/z\}$) $\theta_1 \leq_E \delta$ and $\theta_2 \leq_E \delta$. Thus, $\{\theta_1, \theta_2\}$ is CMU.
- Let $E = \{x + (y + z) = (x + y) + z\} = \mathrm{A}$ (associativity of $+$), $s = x + 1$ and $t = 1 + x$. We have that $CMU = \{\theta_1 = \{1/x\}, \theta_2 = \{1 + 1/x\}, \ldots, \theta_n = \{1 + 1 + \ldots + 1/x\}, \ldots\}$ is an infinite set.

•

E-unification is in general undecidable. A well-known example is given by the Diophantine equations $p_1(x) =_E p_2(x)$, where the polynomials $p_i(x)$ $(i = 1, 2)$ have coefficients in $\mathbf{Z}$ and we search for solutions in $\mathbf{Z}$. This is also known as Hilbert's tenth problem, solved in a negative way in 1970 by Matijasevic.

Many problems on the decidability of E-unification are open. There are several E-unification algorithms for classes of axioms of various kinds. A fairly general case is the following: if the set of axioms E admits an equivalent canonical rewrite system, then E-unification is semi-decidable (i.e. the procedure may not terminate in some cases).
At this point we need to define the notion of 'narrowing'.

8.2 Narrowing

If instead of instantiating a rewrite rule for rewriting a reducible term, such term is unified with the left-hand side of the rule and then the rule is applied, we get a narrowing relation, written $\leadsto$. Let us consider an example.

Example 20: The term $h(f(y, a))$ cannot be reduced with $f(a, x) \longrightarrow g(b)$, but if the subterm $f(y, a)$ is unified with the left-hand side of the rule, the values of y get restricted to the value a and the resulting restricted (narrowed) term $h(f(a, a))$ can be rewritten using the rule to the term $h(g(b))$. •

From the point of view of programming languages, when replacing matching with unification, we move from functional programming towards logic programming (see Section 8.5).

Definition 29 *(narrowing) Let $t \in \mathcal{T}(\Sigma, V)$. Let $p \in Pos'(t)$ and $l \longrightarrow r$ be a rule in R such that $t|_p$ unifies with l, i.e. there exists an mgu σ such that $\sigma(l) = \sigma(t|_p)$. We say that the term t* reduces via narrowing *to the term $t' = \sigma(t)[\sigma(r)]_p = \sigma(t[r]_p)$ and write $t \leadsto_R t'$, where $\leadsto_R$ is the narrowing relation on $\mathcal{T}(\Sigma, V)$ defined by R.*

Let us introduce a fresh new symbol $||$ that does not occur in the signature Σ of the language under consideration. Given $s, t \in \mathcal{T}(\Sigma, V)$, $||(s, t)$ denotes a new term, where narrowing is indifferently applied to either s or t, so that we write the narrowing sequence

$$||(s,t) \leadsto ||(s_1,t_1) \leadsto ||(s_2,t_2) \leadsto \dots$$

Often, reductions by narrowing do not terminate. The reduction graph resulting from the application of the narrowing relation may be very large, thus several optimizations have been developed that do not compromise the completeness of the procedure. Here, we consider:

- *Basic* narrowing
  Narrowing is not applied on a term resulting from a previous application of narrowing.

- *Normal* narrowing
  Narrowing is applied only on terms that are in normal form with respect to the rewrite system R.

Let us first consider the normal narrowing.

8.3 The E-unification procedure

The basic idea of the E-unification procedure defined by Fay, Hullot and Lankford is that in

$$||(s,t) \leadsto ||(s_1,t_1) \leadsto ||(s_2,t_2) \leadsto \ldots \leadsto ||(s_n,t_n)$$

if s_n and t_n are syntactically unifiable, then s and t are E-unifiable by means of the substitution resulting from the composition of the n narrowing unifying substitutions and the syntactic unifier, according to their order of application.

The input to the E-unification procedure is given by the canonical rewrite system R equivalent to the equational theory E and the initial goal $s =_E t$ with the empty substitution id, written $(||(s,t),\ id)$.
Let σ_i the i-th composed substitution of normal narrowing, that is at the i-th step we have $(||(s_i,t_i),\ \sigma_i)$. The steps of the procedure are as follows:

$||(s_i,t_i)$ is in normal form in R and s_i and t_i are syntactically unifiable with mgu θ.
Then, $\theta \circ \sigma_i$ is an E-unifier of s and t.

$||(s_i,t_i)$ is in normal form in R and is not unifiable with the left-hand side of any rule in R.
Then, s_i and t_i are not E-unifiable.

$||(s_i, t_i)$ is in normal form in R and can be reduced via narrowing with a rule in R and substitution σ.
Then, $(||(s_i, t_i), \sigma_i)$ is transformed into $(||(s_{i+1}, t_{i+1}), \sigma \circ \sigma_i)$.

This procedure enumerates the E-unifiers of two terms and terminates if such terms are E-unifiable. Correctness and completeness of the procedure are stated by the following result:

Theorem 6 *Let E be an equational theory that admits a canonical rewrite system R. Let $s, t \in \mathcal{T}(\Sigma, V)$ and consider the auxiliary term $||(s, t)$. If in*

$$||(s,t) \leadsto ||(s_1,t_1) \leadsto ||(s_2,t_2) \leadsto \ldots \leadsto ||(s_n,t_n)$$

$||(s_n, t_n)$ is such that s_n and t_n are syntactically unifiable with mgu θ, then s and t are E-unifiable by means of the substitution resulting from the composition of the n narrowing unifying substitutions and the syntactic unifier θ (according to their order of application).
The set of substitutions computed in this way is a complete but not minimal set of E-unifiers of s and t.

Example 21: Let E be the equational theory:

$$\begin{aligned} +(0, x) &= x \\ +(s(x), y) &= s(+(x, y)) \end{aligned}$$

By applying the completion procedure, we get that the canonical rewrite system equivalent to E is simply given by the two rules obtained by orienting the equations of E from left to right with respect to an rpo based on the precedence $+ > s$:

$$\begin{aligned} +(0, x) &\longrightarrow x \\ +(s(x), y) &\longrightarrow s(+(x, y)) \end{aligned}$$

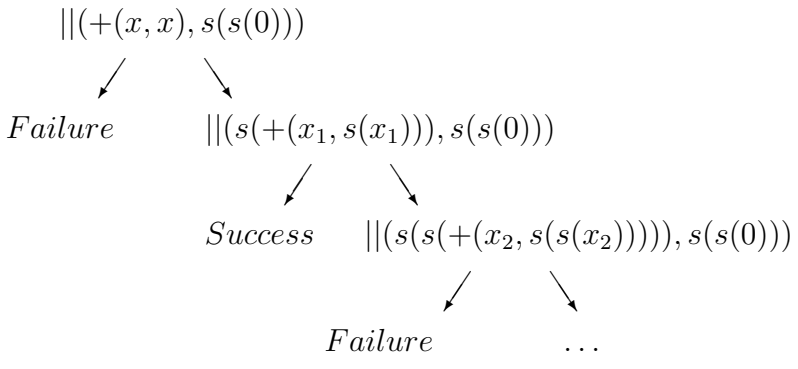
Let $+(x, x) = s(s(0))$ be an equation that we want to solve modulo E using the E-unification procedure based on normal narrowing. The initial term (also referred to as *goal*) is written $||(+(x, x), s(s(0)))$. The terms of the equation are in normal form in R. Starting from the initial goal, the following reductions via narrowing are possible, both at position $p = 1$:

$$\begin{aligned} ||(+(x,x), s(s(0))) &\leadsto ||(0, s(s(0))) \\ ||(+(x,x), s(s(0))) &\leadsto ||(s(+(x_1, s(x_1))), s(s(0))) \end{aligned}$$

by means of, respectively, the first (renamed) rule $+(0, x_1) \longrightarrow x_1$ with mgu $\sigma_1 = \{0/x, 0/x_1\}$, and the second (renamed) rule $+(s(x_1), y_1) \longrightarrow s(+(x_1, y_1))$ with mgu $\sigma_1 = \{s(x_1)/x, s(x_1)/y_1\}$.
The term $||(0, s(s(0)))$ (in normal form in R) is not unifiable with any of the rules in R, and the two terms 0 and $s(s(0))$ are not syntactically unifiable. Hence, we have failure along this path of the narrowing tree. The term $||(s(+(x_1, s(x_1))), s(s(0)))$ (in normal form in R) is unifiable at position $p = 1.1$ with the left-hand side of both rules in R, thus yielding the following reductions via narrowing:

$$\begin{array}{lcl} ||(s(+(x_1, s(x_1))), s(s(0))) & \leadsto & ||(s(s(0)), s(s(0))) \\ ||(s(+(x_1, s(x_1))), s(s(0))) & \leadsto & ||(s(s(+(x_2, s(s(x_2))))), s(s(0))) \end{array}$$

using the rule $+(0, x_2) \longrightarrow x_2$ with mgu $\sigma_2 = \{0/x_1, s(0)/x_2\}$, and the rule $+(s(x_2), y_2) \longrightarrow s(+(x_2, y_2))$ with mgu $\sigma_2 = \{s(x_2)/x_1, s(s(x_2))/y_2\}$.
The two terms of the current goal $||(s(s(0)), s(s(0)))$ syntactically unify with mgu $\theta = id$. By composing the unifying substitutions σ_1, σ_2 and θ, we get the solution $x = s(x_1) = s(0)$. Indeed, if we replace x with $s(0)$ in the initial equation $+(x, x) = s(s(0))$, we have $+(s(0), s(0)) = s(s(0))$ that is true modulo E (the normal form in R of $+(s(0), s(0))$ is $s(s(0))$).
Starting from the other current goal $||(s(s(+(x_2, s(s(x_2))))), s(s(0)))$, it is instead possible to apply further narrowing steps at position $p = 1.1.1$ and then again in more nested positions of the resulting terms without deriving other solutions for the initial equation.
The tree with the narrowing derivations is as follows:



•

The set of E-unifiers is in general complete but not minimal.

In Section 8.2 we have introduced *basic* narrowing, where reductions via narrowing are not applied on those terms that have been derived by means of previous applications of narrowing. This leads to a restriction of the search space in the E-unification procedure without losing correctness and completeness [9]. The procedure is modified by restricting the term positions on which narrowing steps can be applied. The basic positions are a subset of the non-variable positions considered in Definition 29 and are defined as follows. Given the initial term $||(s,t)$, the set of positions pos_0 coincides with $Pos'(||(s,t))$. At the next iterations, given the term $||(s_i,t_i)$ resulting from the application of a narrowing step at position p using rule $l \longrightarrow r$, we have:

$$pos_{i+1} = (pos_i \setminus \{q \mid p \leq q\}) \cup \{p.q \mid q \in Pos'(r)\}$$

Hence, the positions *lower* (or *deeper*) than position p (p included) are replaced by the non-variable positions of the right-hand side (not instantiated) of the rule used at the previous narrowing step.

8.4 Equational theories and E-unification: a summary

This section has not been presented at the lectures of the course, thus is not part of the programme

In the literature the problem of unification modulo an equational theory has been studied for several theories. The most well-known theories are the following:

- $E = \{f(x,y) = f(y,x)\} = C$ (commutativity)
- $E = \{f(x,f(y,z)) = f(f(x,y),z)\} = \mathrm{A}$ (associativity)
- $E = A \cup C = AC$
- $E = \{f(x,x) = x\} = I$ (idempotency)
- $E = A \cup I = AI$
- $E = \{g(x,f(y,z)) = f(g(x,y),g(x,z))\} = D_s$ (left distributivity)
- $E = \{g(f(y,z),x) = f(g(y,x),g(z,x))\} = D_d$ (right distributivity)
- $E = D_s \cup D_d = D$ (distributivity)

- $E = A \cup D = AD$

Given two terms s and t and an equational theory E, let $\mathcal{U}_E(s,t)$ be a complete and minimal set of E-unifiers of s and t. $|S|$ denotes the cardinality of a set S. The unification modulo E is:

- *unitary* if $|\mathcal{U}_E(s,t)| \leq 1$;
- *finitary* if $|\mathcal{U}_E(s,t)| < \omega$;
- *infinitary* if $|\mathcal{U}_E(s,t)| = \omega$;
- *nullary* if $\mathcal{U}_E(s,t)$ does not exist.

We know that syntactic unification ($E = \emptyset$) is unitary (Section 2.5.2). Moreover, we have seen in Example 19 that the theory C of commutativity is finitary and the theory A of associativity is infinitary. The results regarding unification modulo the above equational theories are summarized as follows:

- $E = \emptyset$ unitary;
- C finitary;
- A infinitary;
- AC finitary;
- I finitary;
- AI nullary;
- D_s unitary;
- D_d unitary;
- D infinitary;
- AD infinitary.

For more details on E-unification modulo these equational theories and the related algorithms we refer to [1].

8.5 Comparison with SLD-resolution

This section has not been presented at the lectures of the course, thus is not part of the programme. The students who are familiar with SLD-resolution might find it interesting to read this comparison.

Let us see, through an example, how equations can be solved modulo an equational theory E using the SLD-resolution in Prolog [5, 11].

Example 22: The equational theory E given in Example 21 can be formalized by means of the following Horn clauses:

$$plus(0, x_1, x_1). \tag{4}$$

$$plus(s(x_2), y_2, s(z_2)) \ :- \ plus(x_2, y_2, z_2). \tag{5}$$

The equation $+(x, x) = s(s(0))$ to be solved modulo E becomes the goal:

$:- plus(x, x, s(s(0)))$.

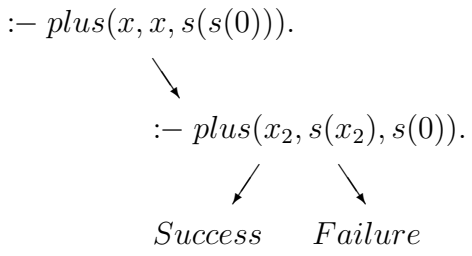
This goal unifies with the head of clause (5) with unifier $\{s(x_2)/x, s(x_2)/y_2, s(0)/z_2\}$. The new goal is

$:- plus(x_2, s(x_2), s(0))$.

that unifies with the head of clause (4) with unifier $\{0/x_2, s(0)/x_1\}$. Thus, we obtain the empty clause

$\Box$.

and by composing the substitutions we get the answer substitution $x = s(x_2) = s(0)$.
The SLD-tree is the following:



•

This example shows that E-unification based on narrowing and SLD-resolution are different approaches to solving equations modulo an equational theory. Narrowing and resolution are inference rules based on the same kind

of mechanism, as they both try to unify a part of the goal (in narrowing it is a subterm of a side of the equation, while in resolution it is a literal) with a part of a rule (in narrowing it is the left-hand side of a rewrite rule, while in resolution it is the head of a clause). Both narrowing and resolution apply the unifying substitution to the whole goal and the whole rule/clause, and derive a new goal by taking the instances of the remaining parts of the goal and of the rule/clause.

The difference between narrowing and resolution is in the fact that narrowing, when trying to unify, considers all possible subterms on which a narrowing step can be applied, while resolution considers the whole literal with all its arguments. It follows that in narrowing all possible choices of subterms are investigated, while in resolution only the paths starting from the application of a resolution step on the selected literal are explored. This implies that narrowing has a much larger search space than resolution and takes more time. In Examples 21 and 22 we can already observe a difference between the search spaces. At the first step narrowing tries (as it is possible) the application of both rules in R. The application of the first rule leads to failure, while the application of the second rule results in a goal on which it is still possible to apply both rules: the first one gives success, whereas the second one leads to an infinite number of failures and the non-termination of the procedure of E-unification based on narrowing. This procedure searches for a complete set of E-unifiers, without recognizing that such a set is represented by the only substitution $\{s(0)/x\}$. Resolution instead, at the first step, can only apply clause (5) and on the resulting goal can apply both (4) and (5). Prolog SLD-resolution applies (4) and terminates with success. The application of (5) derives a goal on which no clauses can be applied, thus yielding failure and termination.

References

[1] F. Baader and T. Nipkow, '*Term Rewriting and All That*', Cambridge University Press, Cambridge, 1998.

[2] B. Buchberger, 'History and Basic Features of the Critical Pair / Completion Procedure', *Journal of Symbolic Computation* **3**, 1987, pp. 3–38.

[3] N. Dershowitz and J.-P. Jouannaud, 'Rewrite Systems', in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, J. van Leeuwen ed., North-Holland, 1990, pp. 243-320.

[4] N. Dershowitz, L. Marcus and A. Tarlecki, 'Existence, uniqueness and construction of rewrite systems', *SIAM Journal of Computing* **17**, 1988, pp. 629–639.

[5] J. H. Gallier, '*Logic for Computer Science: Foundations of Automatic Theorem Proving*', Computer Science and Technology Series, Harper and Row, 1986.

[6] J. Gallier, P. Narendram, D. Plaisted, S. Raatz and W. Snyder, 'An algorithm for finding canonical sets of ground rewriting rules in polynomial time', *Journal of ACM* **40**, 1993, pp. 1–16.

[7] G. Huet, 'Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems', *Journal of ACM* **27**, No. 4, October 1980, pp. 797–821.

[8] G. Huet, 'A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm', *Journal of Computer and System Sciences* **23**, 1981, pp. 11–21.

[9] J.-M. Hullot, 'Canonical Forms and Unification', in Proceedings of $5th$ CADE, Lecture Notes in Computer Science 87, Springer-Verlag, 1980, pp. 318–334.

[10] P. Inverardi, M. Nesi and M. Venturini Zilli, 'Sistemi di Riscrittura per Termini del Prim'Ordine', Technical Report No. 35, Dipartimento di Matematica Pura e Applicata, Università degli Studi di L'Aquila, July 1999.

[11] J. W. Lloyd, '*Foundations of Logic Programming*', 2nd edition, Springer, 1987.

[12] M. Nesi and M. Venturini Zilli, 'Sistemi di riduzione astratti', Research Report SI-98/06, Facoltà di Scienze MM. FF. NN., Università degli Studi di Roma "La Sapienza", March 1998.

[13] A. Sattler-Klein, 'About changing the ordering during Knuth-Bendix completion', *STACS 94, Symposium on Theoretical Aspects of Computer Science* **15**, 1994, pp. 175–186.

[14] W. Snyder, 'A fast algorithm for generating reduced ground rewriting systems from a set of ground equations', *Journal of Symbolic Computation* **15**, 1993, pp. 415–450.

[15] The C*i*ME Rewrite Tool, in `http://cime.lri.fr/`.

[16] ELAN, in `http://elan.loria.fr/`.

[17] Larch Prover, in `http://www.sds.lcs.mit.edu/spd/larch/`.

[18] The Maude System, in `http://maude.cs.uiuc.edu/`.